ELSEVIER

Theoretical Computer Science 164 (1996) 13–28

Theoretical Computer Science

Anomalous learning helps succinctness ☆

John Case [a], Sanjay Jain [b], Arun Sharma [c,*]

[a] *Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA*
[b] *Department of Information Systems and Computer Science, National University of Singapore, Singapore 0511, Singapore*
[c] *School of Computer Science and Engineering, University of New South Wales, Sydney, NSW, 2052, Australia*

Received March 1994; revised April 1995
Communicated by G. Ausiello

Abstract

It is shown that allowing a *bounded* number of anomalies (mistakes) in the *final* programs learned by an algorithmic procedure can considerably "succinctify" those final programs. Naturally, only those contexts are investigated in which the presence of anomalies is not actually required for successful inference (learning). The contexts considered are certain infinite subclasses of the class of characteristic functions of finite sets. For each finite set D, these subclasses have a finite set containing D. This latter prevents the anomalies from wiping out all the information in the sets featured in these subclasses and shows the context to be fairly robust. Some of the results in the present paper are shown to be provably more constructive than others.

The results of this paper can also be interpreted as facts about succinctness of coding finite sets, which facts have interesting consequences for learnability of decision procedures for finite sets.

1. Introduction

A subject encounters data about a concept. Based on this finite amount of data, the subject conjectures a hypothesis about the concept. Availability of more data may cause the subject to change its hypothesis. The subject is said to learn the concept just in case the sequence of hypotheses conjectured stabilizes to a fixed hypothesis and this final hypothesis is a correct representation of the concept.

Aspects of learning by humans may be cast in the above framework. It may also be argued that the hypotheses inferred by humans tend to display the following

☆ A preliminary version of this paper appeared in S. Arikawa, S. Goto, S. Ohsuga and T. Yokomori, editors, *Algorithmic Learning Theory*, Proceedings of the First International Workshop, Tokyo, Japan, pages 282-288, Japanese Society for Artificial Intelligence, October 1990 (Reprinted by Ohmsa – Springer Verlag).

* Corresponding author. Email: arun@cse.unsw.edu.au.

SSDI 0304-3975(95)00154-9

characteristics:

- they are only an approximate representation of the concept;
- they are likely to be succinct.

Many studies in inductive inference have addressed the above two issues separately. For example in the context of learning computer programs for computable functions, Case and Smith [6] considered the situation in which the final program is allowed to make a finite, but bounded, number of mistakes in computing the function. For the same learning task, Royer [22] and Fulk and Jain [12] consider models in which the final program is allowed to make infinitely many mistakes in computing the function, but the "density" of these mistakes is bounded. The subject of succinctness for function identification has also been considered by many authors, for example, Freivalds [10], Freivalds and Kinber [11], Kinber [15], Chen [7, 8], Case et al. [4], Jain and Sharma [14], and Case et al. [5].

However, while the above studies motivate each of inferring approximate solutions and inferring nearly minimal solutions, none of these studies have reflected on why humans appear to have a predilection for inferring solutions which are *simultaneously* approximate and succinct. The present paper begins to address this issue by presenting some preliminary results suggesting that, in many learning situations, approximate hypotheses may turn out to be far more succinct than the minimal size, exact hypothesis.

The general framework of our results is identification in the limit of computer programs for computable functions from their graphs. Specifically, our results are about learning programs for the characteristic functions of finite sets, i.e., about identifying decision procedures for finite sets.

Case and Smith [6] (also see [3]) showed that learners allowed to converge to programs that make a finite number of mistakes can learn strictly larger collections of functions than those learners that are required to converge to exact programs. Thus, there are collections of functions that can be learned only if anomalies are allowed in the final inferred program. The collection of characteristic functions of finite sets can be trivially learned without resorting to learners that commit anomalies in the final inferred program. Nonetheless, we show that there are nontrivial collections of characteristic functions of finite sets for which allowing an error in the final program yields tremendous savings in the size of the final program compared with the minimal size exact program.

We now note a crucial property of the collections of characteristic functions of finite sets considered in the present paper. A 1-error program for the collection of characteristic functions of singleton sets can be trivially identified by a machine that simply outputs a decision procedure for the empty set. Clearly, demonstrating the size advantages of allowing errors in the final program *for such example classes* is not at all interesting since some 1-error programs for deciding *singleton* sets are quite useless. For this reason, our results are about those collections that contain, for each finite set D, the characteristic function of another finite set containing D. Such collections are referred to as *beefy*. Clearly, a 1-error program for each member of a *beefy* class would be useful since most elements of such a class would be quite far from characteristic functions of singleton sets.

We now informally describe some of our results. To this end, it is useful to go over some recursion theoretic terminology. Programs can be treated as numbers and these numbers in turn can be treated as corresponding program size measures (see [2]). **Fin** denotes the set of all characteristic functions of finite sets. For any total (possibly noncomputable) function h, we say that program p is *h-more succinct than* a program q just in case $h(p) < q$, i.e., the magnification of p by h is still not as large as q.

Our results are essentially of two kinds. One kind, provably less constructive than the other, holds for any, not necessarily computable, "succinctness factor" h. The other holds for a much more limited class of h. Proofs of the former kind of result (e.g., the proof of Theorem 2) feature counting arguments, and our proofs of the latter kind (e.g., the proof of Theorem 5) involve more constructive, recursion theoretic arguments. The latter provide the advantage of giving us an algorithmic handle on the classes for which we prove existence while the former do not.

Corollary 1, which is a special case of Theorem 2 and which can be proved using a counting argument, implies that for any total (noncomputable) function h, there is a beefy collection of functions $\mathscr{C}$ such that there exists a learning machine $\mathbf{M}_1$ satisfying the following.

- $\mathbf{M}_1$ identifies a 1-error program in the limit for any member of **Fin**, and
- $\mathbf{M}_1$'s final conjecture on any $f \in \mathscr{C}$ is h-more succinct than the final conjecture of any machine $\mathbf{M}$ on f that identifies in the limit a 0-error program for each member of $\mathscr{C}$.

Theorem 3 shows that, in most cases, the beefy class $\mathscr{C}$ in Corollary 1 is not a recursively enumerable class. Hence, the natural and convenient algorithmic handle of being an r.e. class is just not available for the classes whose existence we show. We consider, then, a slightly less natural and convenient algorithmic handle on our classes which *is* available. Theorem 5 provides a constructive existence featuring classes about which we have *some* algorithmic knowledge. For expository convenience, we discuss here only a special case of this result, Corollary 2. We first introduce a notation: a 1-*extension* of a partial computable function ψ is a total function f such that, for each x in domain of ψ, $f(x) = \psi(x)$ and for each x not in domain of ψ, $f(x) = 1$.

Corollary 2 shows that, for any *limiting recursive* h [25–28][1], from an index of a program that computes h in the limit, we can *algorithmically*, find an r.e. index for a class of partial recursive functions whose 1-extension, $\mathscr{C}$, is beefy and such that there exists a machine $\mathbf{M}_1$ that satisfies the following:

- $\mathbf{M}_1$ identifies in the limit a 1-error program for each member of **Fin**, and
- $\mathbf{M}_1$'s final conjecture on each $f \in \mathscr{C}$ is h-more succinct than the final conjecture of *any* machine $\mathbf{M}$ on f that identifies in the limit a 0-error program for each member of $\mathscr{C}$.

[1] Intuitively, *limiting-recursive functions* are (total) functions computed by programs which do not give correct output until after some unspecified but finite number of trial outputs. See the formal definition near the end of Section 2.

Unfortunately, as implied by Theorem 4, Corollary 2 does not hold for any arbitrary h.

Finally, an alternative interpretation of the results presented in the present paper is that there are "non trivial" finite sets for which a "slightly" anomalous decision procedure is arbitrarily more succinct than an exact decision procedure. Thus, the present paper may be viewed as containing results about coding finite sets presented learning theoretically.

We now proceed formally.

2. Notation

Recursion-theoretic concepts not explained below are treated in [21]. N denotes the set of natural numbers, $\{0,1,2,3,\ldots\}$. N^+ denotes the set of positive integers, $\{1,2,3,\ldots\}$. Unless otherwise specified, a, e, i, j, k, l, m, n, q, r, s, u, w, x, y, z, with or without decorations (decorations are subscripts, superscripts and the like), range over N. $\emptyset$, $\in$, $\subseteq$, $\subset$, $\supseteq$, $\supset$, denote empty set, element of, subset, proper subset, superset and proper superset, respectively. A, S, P, with or without decorations, range over sets. D, with or without superscripts, ranges over finite sets. Fix a *canonical* indexing $D_0, D_1, \ldots$ of the finite sets [21]. For the ease of using finite sets as arguments to recursive functions, we identify finite sets with their canonical indices. For $n_1 < n_2$, $[n_1 .. n_2]$ denotes the set $\{x \mid n_1 \leqslant x \leqslant n_2\}$. $\mathrm{card}(S)$ denotes the cardinality of S. $\max(S)$ and $\min(S)$ denote the maximum and minimum of set S, respectively. $\max(S) = \infty$, if $\mathrm{card}(S)$ is infinite. By convention $\min(\emptyset) = \infty$ and $\max(\emptyset) = 0$.

f, g, h and F, with or without decorations, range over total functions. η and θ, with or without decorations, range over (possibly) partial functions. $\mathscr{R}$ denotes the class of all *recursive* functions, i.e., total computable functions with arguments and values from N. For $n \in N^+$, $\mathscr{R}^n$ denotes the class of total recursive functions of n variables. $\mathscr{C}$ and $\mathscr{S}$ range over subsets of $\mathscr{R}$. For *partial* functions η and θ, $\eta =^a \theta$ means that $\mathrm{card}(\{x \mid \eta(x) \neq \theta(x)\}) \leqslant a$. $\mathrm{domain}(\eta)$ and $\mathrm{range}(\eta)$ denote domain and range of partial function η.

Suppose that $\mathsf{E}(x_1,\ldots,x_n)$ is an expression with $x_1,\ldots,x_n$ as its only free variables. Then $\lambda x_1,\ldots,x_n.\mathsf{E}(x_1,\ldots,x_n)$ denotes the function that maps $(x_1,\ldots,x_n)$ to the value $\mathsf{E}(x_1,\ldots,x_n)$ [9, 21]. For example, $\lambda x.x+1$ denotes the function that maps x to $x+1$ and $\lambda x, y.x+y$ denotes the function that maps (x,y) to $x+y$.

$\lambda x, y.\langle x, y\rangle$ denotes a fixed pairing function (a recursive, bijective mapping: $N \times N \to N$) [21]. $\lambda x, y.\langle x, y\rangle$ and its inverses are useful to simulate the effect of having multiple argument functions. $\langle \cdot, \cdot \rangle$ can be extended to encoding of multiple arguments in a natural way.

φ denotes a fixed acceptable programming system for the partial computable functions: $N \to N$ [20, 21, 17]. φ_i denotes the partial computable function (of one argument) computed by the ith program in the φ system. Hereinafter *program* i refers to the ith program in the φ programming system. We shall speak of programs and numerical

names or codes for programs interchangeably; sometimes these numerical names are referred to as *indices*. In some contexts p, range over natural numbers thought of, in those contexts, as programs. In other contexts p ranges over total functions in which the range of p is thought of as a set of programs. Occasionally, we will write $\varphi_i(x, y)$ as an abbreviation for $\varphi_i(\langle x, y\rangle)$. $W_i = \text{domain}(\varphi_i)$. W_i is, then, the recursively enumerable (r.e.) set accepted by φ-program i. Φ denotes an arbitrary Blum complexity measure associated with acceptable programming system φ; such complexity measures exist for any acceptable programming system [1, 17].

For any set $P \subseteq N$, $\mathscr{C}_P$ denotes the class of partial recursive functions $\{\varphi_p \mid p \in P\}$. $MinProg(f)$ denotes $\min(\{p \mid \varphi_p = f\})$. $MinProg_i(f)$ denotes $\min(\{p \mid \varphi_p =^i f\})$.

χ_A denotes the *characteristic function of* A, the function which is 1 on A and 0 off A. **Fin** denotes the class of characteristic functions of all finite sets, i.e., **Fin** $= \{\chi_A \mid \text{card}(A) < \infty\}$. We fix a recursive function *Prog* such that, for all finite sets D, $Prog(D)$ is a program (in the φ-system) for χ_D. The s-m-n theorem guarantees the existence of *Prog* [21].

A total function h is called *limiting recursive* just in case there exists a $g \in \mathscr{R}^2$ such that for each $n \in N$, $h(n) = \lim_{t\to\infty} g(n, t)$.

The quantifiers "$\overset{\infty}{\forall}$" and "$\overset{\infty}{\exists}$" mean "for all but finitely many" and "there exist infinitely many", respectively.

3. Preliminaries

A machine that learns a computer program for a computable function from its graph is presented with initial fragments of the graph. We first formalize the notion of segment.

For $m \in N$, let N_m denote the set $\{x \mid x < m\}$. We define a *segment* to be a mapping from N_m, for some $m \in N$, into N. We let SEG denote the set of all segments. We let σ and τ, with or without decorations, range over SEG . $|\sigma|$ denotes the length of σ.

For a (partial) function η, which is defined for all $x < n$, $\eta[n]$ denotes the finite segment σ, of length n, such that, for $x < n$, $\sigma(x) = \eta(x)$.

Definition 1 (*Gold* [13]). An *inductive inference machine* (abbreviated: IIM) is an algorithmic device that computes a mapping from SEG into N.

We let **M**, with or without decorations, range over inductive inference machines. In Definitions 3 and 4 we spell out what, for this paper, it can mean for an inductive inference machine to (reasonably) successfully learn (in the limit) a program for a recursive function. The first of those definitions, Definition 3, has to do with a criterion of "perfect" success. The latter one, Definition 4 has to do with success criteria which permit a *bounded* number of mistakes. But, first, in Definition 2, we introduce a technical notion which says what it means for an inductive inference machine to converge on a function.

Definition 2 (*Blum and Blum* [3] *and Case and Smith* [6]). Suppose $\mathbf{M}$ is an inductive inference machine and $f \in \mathscr{R}$. $\mathbf{M}(f)\downarrow$ (read: $\mathbf{M}(f)$ *converges*) $\iff (\exists p)(\overset{\infty}{\forall}\, n)$ $[\mathbf{M}(f[n]) = p]$. If $\mathbf{M}(f)\downarrow$, then $\mathbf{M}(f)$ is defined $=$ the unique p such that $(\overset{\infty}{\forall}\, n)$ $[\mathbf{M}(f[n]) = p]$; otherwise $\mathbf{M}(f)$ is said to diverge (written: $\mathbf{M}(f)\uparrow$).

Definition 3 (*Gold* [13], *Blum and Blum* [3] *and Case and Smith* [6]).
(a) $\mathbf{M}$ **Ex**-*identifies* f (written: $f \in \mathbf{Ex}(\mathbf{M})$) $\iff (\exists p \mid \varphi_p = f)[\mathbf{M}(f)\downarrow = p]$.
(b) $\mathbf{Ex} = \{\mathscr{S} \subseteq \mathscr{R} \mid (\exists \mathbf{M})[\mathscr{S} \subseteq \mathbf{Ex}(\mathbf{M})]\}$.

Definition 4 (*Case and Smith* [6]). Let $a \in N$.
(a) $\mathbf{M}$ $\mathbf{Ex}^a$-*identifies* f (written: $f \in \mathbf{Ex}^a(\mathbf{M})$ $\iff (\exists p \mid \varphi_p =^a f)[\mathbf{M}(f)\downarrow = p]$.
(b) $\mathbf{Ex}^a = \{\mathscr{S} \subseteq \mathscr{R} \mid (\exists \mathbf{M})[\mathscr{S} \subseteq \mathbf{Ex}^a(\mathbf{M})]\}$.

Clearly $\mathbf{Ex}^0 = \mathbf{Ex}$. The following theorem (Theorem 1) due to Case and Smith [6] describes a strict hierarchy of identification classes based on the number of anomalies allowed in the final program.

Theorem 1 (Case and Smith [6]). $\mathbf{Ex}^0 \subset \mathbf{Ex}^1 \subset \mathbf{Ex}^2 \subset \cdots$

Definition 5 (*Blum* [2], *Meyer* [18], *Meyer and Bagchi* [19] *and Royer and Case* [23, 24]). A φ-program p is *h-more succinct* than a φ-program $q \iff h(p) < q$, i.e., the "magnification of p by h" is still not as large as q.

In contexts in which one machine $\mathbf{M}_1$ in one sense identifies a function f and another machine $\mathbf{M}_2$ in a possibly different sense also identifies f, we sometimes informally speak of $\mathbf{M}_1$ *identifying* f *h-more succinctly than* $\mathbf{M}_2$ $\iff$ the final program witnessing the identification of f by $\mathbf{M}_1$ is h-more succinct than the final program witnessing the identification of f by $\mathbf{M}_2$.

We next formally record the definition of a beefy class.

Definition 6. $\mathscr{C}$ is *beefy* $\iff$ $[[\mathscr{C} \subseteq \mathbf{Fin}] \wedge [(\forall D, \text{finite})(\exists D', \text{finite} \mid D' \supseteq D)[\chi_{D'} \in \mathscr{C}]]]$.

4. Results

Our results featuring the existence of beefy collections of characteristic functions of finite sets are essentially of two kinds. One kind, provably less constructive than the other, holds for any, not necessarily computable, "succinctness factor" h. The other holds for a much more limited class of h. Proofs of the former kind of results feature counting arguments, a simple special case of which is illustrated in the proof of Proposition 1. The possibility of getting at least some theorems, of the kind found in this paper, by counting arguments was first suggested to the first author by Dana

Angluin. Our proofs of the latter, more constructive results involve recursion theoretic arguments rather than counting arguments.[2]

Proposition 1 (K.-J. Chen, unpublished). *Suppose h is any total function. Suppose g is recursive and e is a program for g. Then there are infinitely many recursive functions f such that $f =^1 g$ and $(\forall i)[\varphi_i = f \Rightarrow i > h(e)]$.*

Proof. Let $\mathscr{C} = \{f \in \mathscr{R} \mid f =^1 g\}$. $\mathscr{C}$ is the set of all recursive functions which differ from f at no more than one argument. Clearly, card($\mathscr{C}$) is ∞. Now, there are only finitely many programs $\leqslant h(e)$. Let $\mathscr{S} = \{f \in \mathscr{C} \mid (\exists i \leqslant h(e))[\varphi_i = f]\}$. $\mathscr{S}$ is the set of all such functions $f \in \mathscr{C}$ for which there is a φ-program $\leqslant h(e)$. Clearly, card($\mathscr{S}$) is finite. Thus, card($\mathscr{C} - \mathscr{S}$) is ∞ and $(\forall f \in (\mathscr{C} - \mathscr{S}))(\forall i)[\varphi_i = f \Rightarrow i > h(e)]$. □

We present our results in three stages.

1. In Section 4.1, we show that there are beefy classes for which allowing anomalies in the final program can result in convergence to arbitrarily succinct programs.

2. Section 4.2 looks at the nature of such beefy classes. In particular, it is shown that such classes cannot be r.e.

3. The above negative result implies that there is no straightforward description (like an algorithmic enumerator of programs) for functions in these beefy classes. Results in Section 4.3 are aimed at finding a "partial" description of such beefy classes.

4.1. Nonconstructive succinctness results

Theorem 2 is our first result showing the succinctness advantage of allowing anomalies in the final program. For ease of discussion, we first present a special case of Theorem 2. This result, Corollary 1, says that, for *any* total function h, there exists a beefy $\mathscr{C}$ such that there is a machine $\mathbf{M}_1$ that $\mathbf{Ex}^1$-identifies **Fin**, the entire class of characteristic function of finite sets, and $\mathbf{M}_1$ $\mathbf{Ex}^1$-identifies each function in $\mathscr{C}$ h-more succinctly than *any* machine that **Ex**-identifies each function in $\mathscr{C}$. Of course, the entire class of characteristic functions of *finite* sets is trivially **Ex**-identifiable. The point of Corollary 1 is that even in cases where one does not have to allow a possible anomaly in the final program to achieve success [6], allowing a possible anomaly can, nonetheless, considerably cut down on the size of the final programs inferred! Beefiness is important because, then, one anomaly is not so serious: if the sets were, for example, singletons, a program with one anomaly to decide them might be totally useless.

Corollary 1. $(\forall$ *total* $h)(\exists$ *a beefy* $\mathscr{C})(\exists \mathbf{M}_1)[\mathbf{Fin} \subseteq \mathbf{Ex}^1(\mathbf{M}_1) \wedge (\forall \mathbf{M} \mid \mathscr{C} \subseteq \mathbf{Ex}(\mathbf{M}))(\forall f \in \mathscr{C})[\mathbf{M}(f) > h(\mathbf{M}_1(f))]]$.

[2] Our results based on counting arguments could also be obtained by Kolmogorov complexity arguments [16]; however, our more constructive results, which provide an algorithmic handle on the classes whose existence we prove, likely cannot.

We now discuss the general result.

Suppose h is any arbitrary total function and $n \in N$. A general question, then, would be whether there exists a beefy $\mathscr{C}$ and n machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n$ such that, for $1 \leqslant k \leqslant n$, machine $\mathbf{M}_k$ $\mathbf{Ex}^k$-identifies **Fin**, and for any $k < n$, machine $\mathbf{M}_{k+1}$ $\mathbf{Ex}^{k+1}$-identifies each $f \in \mathscr{C}$, h-more succinctly than any machine that $\mathbf{Ex}^k$-identifies each $f \in \mathscr{C}$. It turns out, thanks to Theorem 2 just below, that such is the case.

Theorem 2. $(\forall$ *total* $h)(\forall n)(\exists$ *a beefy* $\mathscr{C})(\exists \mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n)$ $[(\forall k \mid 1 \leqslant k \leqslant n)[\mathbf{Fin} \subseteq \mathbf{Ex}^k(\mathbf{M}_k)] \wedge (\forall f \in \mathscr{C})(\forall k < n)[MinProg_k(f) > h(\mathbf{M}_{k+1}(f))]]$.

Proof. First we describe machine $\mathbf{M}_j$, $1 \leqslant j \leqslant n$.

begin $\{\mathbf{M}_j\}$
on input σ:
  let $D = \{x \mid \sigma(x) \neq 0\}$;
  if card$(D) < j$ **then**
    let $S = D$
  else
    let S be the set consisting of j largest elements in D
  endif;
  output $Prog(D - S)$
end $\{\mathbf{M}_j\}$

It is easy to verify that, for $1 \leqslant j \leqslant n$, $\mathbf{M}_j$ $\mathbf{Ex}^j$-identifies **Fin**.

For each $i \in N$, we now define f_i, a characteristic function of a finite set. For each $i \in N$, let S_i denote the finite set $[0..i]$. To facilitate the description of f_i, we introduce n numbers $a_{i,1}, a_{i,2}, \ldots, a_{i,n}$ defined as follows. We select $a_{i,r}$'s in sequence as follows: for $r \leqslant n$, suppose $a_{i,1}, \ldots, a_{i,r-1}$ have already been selected. We chose $a_{i,r}$ to be the least number, a, greater than $\max(S_i \cup \{a_{i,x} \mid x < r\})$ such that each partial function computed by a program $\leqslant h(Prog(S_i \cup \{a_{i,x} \mid x < r\}))$ is either, non 0 for infinitely many inputs, or, is 0 for all inputs $\geqslant a$.

Formally, for $1 \leqslant r \leqslant n$, $a_{i,r} = \min(\{a > \max(S_i \cup \{a_{i,l} \mid 1 \leqslant l < r\}) \mid (\forall w \leqslant h(Prog(S_i \cup \{a_{i,l} \mid 1 \leqslant l < r\})))[[card(y \mid \varphi_w(y) \neq 0) = \infty] \vee [(\forall z \geqslant a)[\varphi_w(z) = 0]]]\})$.

Let $f_i = \chi_{(S_i \cup \{a_{i,l} \mid 1 \leqslant l \leqslant n\})}$. We take $\mathscr{C} = \{f_i \mid i \in N\}$. It is easy to verify that $\mathscr{C}$ is beefy and $(\forall f \in \mathscr{C})(\forall k < n)[MinProg_k(f) > h(\mathbf{M}_{k+1}(f))]$. □

4.2. *Beefy classes witnessing succinctness are not r.e.*

A natural question is to investigate the nature of the candidate beefy classes witnessing succinctness results in the previous section. In particular, it would be nice if we could have an algorithmic handle on them. Unfortunately, it turns out that the candidate class $\mathscr{C}$ in Theorem 2 above, in many cases, is not a recursively enumerable class [21]: Theorem 3 implies that there exists a recursive, monotone increasing, function h such that, for all beefy *recursively enumerable classes* $\mathscr{C}$, there is a machine $\mathbf{M}$ that

Ex-identifies $\mathscr{C}$ and, for infinitely many f in $\mathscr{C}$, the final program output on f by any machine that $\mathbf{Ex}^1$-identifies $\mathscr{C}$ is *not* h-more succinct than $\mathbf{M}(f)$.

Theorem 3. ($\exists$ *a recursive, monotone increasing function* h)($\forall$ *beefy r.e. classes* $\mathscr{C}$) ($\exists\mathbf{M}$)[$[\mathbf{Fin} \subseteq \mathbf{Ex}(\mathbf{M})] \wedge (\overset{\infty}{\exists} f \in \mathscr{C})[\mathbf{M}(f) \leqslant h(MinProg_1(f))]]$.

Theorem 3 follows from Lemmas 1 and 2. We first introduce a technical notion.

Definition 7. A set of programs P is called 2-*discrete* $\iff$ $(\forall i,j \in P \mid i \neq j)$ $[card(\{x \mid \varphi_i(x) \neq \varphi_j(x)\}) > 2]$.

Lemma 1. *For each* $k \in N$ *such that* $\mathscr{C}_{W_k}$ *is beefy, we can algorithmically find a* k' *satisfying the following three conditions*:

(a) $W_{k'} \subseteq W_k$;

(b) $\mathscr{C}_{W_{k'}}$ *is beefy*;

(c) $W_{k'}$ *is 2-discrete*;

(d) $(\forall x)(\overset{\infty}{\forall} j \in W_{k'})[\varphi_j(x) = 1]$.

Proof. We give an informal description for enumerating $W_{k'}$ from an enumeration of W_k.

Let $W_{k'}^s$ denote the finite subset of $W_{k'}$ enumerated before Stage s.

Thus $W_{k'}^0 = \emptyset$. Go to Stage 0;

begin {Stage s}

Search for a $j \in W_k$ such that $[0..s] \subseteq \{x \mid \varphi_j(x) = 1\}$ and, for all $l \in W_{k'}^s$, $\text{card}(\{x \mid \varphi_j(x)\downarrow \neq \varphi_l(x)\downarrow\}) > 2$.

If and when such a j is found enumerate j in $W_{k'}$. Go to Stage $s+1$.

end {Stage s}

It is easy to verify that the above construction for the enumeration of $W_{k'}$ satisfies the lemma. □

Lemma 2. ($\exists h \in \mathscr{R}$)($\forall j \mid W_j$ *is 2-discrete,* $\mathscr{C}_{W_j}$ *is beefy and* $(\forall x)(\overset{\infty}{\forall} l \in W_j)[\varphi_l(x) = 1]$)($\exists\mathbf{M}$)[$[\mathbf{Fin} \subseteq \mathbf{Ex}(\mathbf{M})] \wedge (\overset{\infty}{\exists} f \in \mathscr{C}_{W_j})[\mathbf{M}(f) \leqslant h(MinProg_1(f))]]$.

Proof. Below by implicit use of a suitably padded version of the s-m-n theorem, we define an infinite sequence of φ-programs $\{p(i) \mid i \in N\}$, such that p is a recursive, strictly increasing function, but first we show how to define h in terms of p.

We divide the collection of φ-programs $\{p(i) \mid i > 0\}$ into groups. The mth group consists of $m+2$ programs. The 0th group is $\{p(0), p(1)\}$. The 1st group is $\{p(2), p(3), p(4)\}$ and so on. Formally, for $m \in N$, the mth group is $\{p(k + \frac{m\cdot(m+3)}{2}) \mid k < m+2\}$.

We define h in such a way that, for $m \in N$, $h(m)$ is greater than the largest index in the mth group. This is achieved by setting, for each m, $h(m) = p(\frac{(m+1)\cdot(m+4)}{2})$.

We assume without loss of generality that the pairing function $\langle \cdot, \cdot \rangle$ is such that $(\forall j,k)[\langle j,k\rangle < \langle j,k+1\rangle]$.

For all $k,j,l \in N$ such that $k < \langle j,l\rangle + 2$, we let

$$F(k,j,l) = p(k + \frac{\langle j,l\rangle \cdot (\langle j,l\rangle + 3)}{2}).$$

Then $F(k,j,l)$ denotes the kth program in the $\langle j,l\rangle$th group of programs mentioned above. Groups numbered $\langle j,l\rangle$, $l \in N$, will be used to handle W_j. Clearly, from any j, we can uniformly algorithmically create a 1–1 enumeration of W_j. Do so in a fixed way. $\varphi_{F(k,j,l)}$ is defined to be the (partial) function computed by the $(\sum_{r=0}^{r=l-1}(\langle j,r\rangle + 2) + k)$th program in the 1–1 listing of W_j if such a program exists; otherwise, $\varphi_{F(k,j,l)}$ is undefined on all inputs. For j satisfying the hypothesis of Lemma 2, for $l \in N$, the purpose of $\langle j,l\rangle$th group is to contain at least one program for a characteristic function in $\mathscr{C}_{W_j}$ which is not computed by any program in the $\langle j,r\rangle$th groups, $r < l$, such that this new characteristic function can be succinctly and exactly identified.

If W_j is 2-discrete and $\mathscr{C}_{W_j}$ is beefy, then functions in $\bigcup_{r\in N}\bigcup_{k\leqslant\langle j,r\rangle+1}\{\varphi_{F(k,j,r)}\}$ are pairwise different from each other at more than 2 arguments. Hence, in this case, for each $r \in N$, there exists a $k < \langle j,r\rangle + 2$, such that $MinProg_1(\varphi_{F(k,j,r)}) > \langle j,r\rangle$ and $h(\langle j,r\rangle) > F(k,j,r)$.

Suppose j is as described in the hypothesis of Lemma 2. Then, for each i such that

$$p(i) \in \bigcup_{r\in N} \bigcup_{k\leqslant\langle j,r\rangle+1} \{F(k,j,r)\}, \tag{1}$$

we have that $\varphi_{p(i)} \in \mathbf{Fin}$. We give below the description of a machine $\mathbf{M}$ which $\mathbf{Ex}$-identifies $\mathbf{Fin}$ in such a way that, for all i satisfying (1) above, $\mathbf{M}$ converges to $p(i)$ on input $\varphi_{p(i)}$. Such an $\mathbf{M}$ clearly witnesses the claim of Lemma 2.

begin {**M**}
on input $f[n]$:
  let g_i denote the function computed by the i-th program in the fixed 1-1 algorithmic enumeration of W_j;
  if $(\exists i < n)(\forall x < n)[g_i(x) = f(x)]$
    then
    output $F(k,j,l)$, where $l = \max(\{m \mid [\sum_{r=0}^{r=m-1}(\langle j,r\rangle + 2) \leqslant i]\})$
    and $k = i - \sum_{r=0}^{r=l-1}(\langle j,r\rangle + 2)$
    else
    output $Prog(\{x < n \mid f(x) = 1\})$
    endif
  end {**M**}

It is easy to verify that $\mathbf{M}$ behaves as in the claim of Lemma 2. □

4.3. Attempts at constructivity

One of our central concerns is *constructivity* for the classes of characteristic functions of finite sets for which allowing anomalies results in identification by strictly more succinct programs. We discuss our motivation for this concern.

The main result of the previous section implies that the beefy classes witnessing succinctness results may not be r.e. If these classes were r.e. then we could hope for a "good" algorithmic descriptor of the classes, i.e. we could have a computer program that enumerated a program for each member of the class. In the event of such a descriptor not being possible, we attempt to find an alternative, if not as good, descriptor.

To this end, we introduce the following notion.

Definition 8. For $P \subseteq N$, we define $\mathscr{C}^1(P) = \{f \mid (\exists j \in P)(\forall x)[\ [\varphi_j(x)\downarrow \Rightarrow f(x) = \varphi_j(x)] \wedge [\varphi_j(x)\uparrow \Rightarrow f(x) = 1]]\}$. We say that $\mathscr{C}^1(P)$ is *1-extension* of the class of partial recursive functions $\mathscr{C}_P$.

To understand the above definition, let $P \subseteq N$ be given. Recall that $\mathscr{C}_P$ denotes the collection of partial recursive functions η such that P contains an index for η. 1-extension of a partial recursive function θ is a total function that agrees with θ on those arguments on which θ is defined, and is 1 on those arguments on which θ is undefined. Then, $\mathscr{C}^1(P)$ is essentially the collection of 1-extensions of partial recursive functions in $\mathscr{C}_P$.

Here is what we are able to say about the description of some of the beefy classes witnessing succinctness results in this paper. If the succinctness factor is limiting recursive, then

- these beefy classes can be shown to be 1-extension of an r.e. class of partial recursive functions;
- and one can algorithmically find an acceptor (or, equivalently a generator) for the r.e. class from a program that computes the succinctness factor in the limit.

Somewhat more precisely, Corollary 2 (to Theorem 5) just below implies that, for any *limiting recursive* h [25–28], from an index, i, of a program that computes h in the limit, we can *algorithmically*, via a recursive function g, find an r.e. index $g(i)$ for a class of partial recursive functions whose 1-extension $\mathscr{C}^1(W_{g(i)})$ is beefy and there exists a machine $\mathbf{M}_1$ that $\mathbf{Ex}^1$-identifies **Fin** and $\mathbf{M}_1$ $\mathbf{Ex}^1$-identifies each $f \in \mathscr{C}^1(W_{g(i)})$ h-more succinctly than *any* machine that **Ex**-identifies $\mathscr{C}^1(W_{g(i)})$.

Corollary 2. $(\exists g \in \mathscr{R})(\forall i, h \mid h = \lambda x.\lim_{s\to\infty} \varphi_i(x,s)$ *is total*$)[[\mathscr{C}^1(W_{g(i)})$ *is beefy*$] \wedge (\exists \mathbf{M}_1)[\mathbf{Fin} \subseteq \mathbf{Ex}^1(\mathbf{M}_1) \wedge (\forall f \in \mathscr{C}^1(W_{g(i)}))[MinProg(f) > h(\mathbf{M}_1(f))]]]$.

A proof of Corollary 2 can be obtained by a moving anomaly marker construction and is a special case of the general version, Theorem 5, presented later in the section. Such constructions were first employed in [6].

However, before we present the general case of the result, it is worth noting that for *arbitrary* total function h, there need *not* be an r.e. index for a class of partial

recursive functions whose 1-extension is beefy and for which allowing one anomaly results in more succinct identification. This is implied by the next theorem.

Theorem 4. $(\forall n)(\exists \text{ total } h)(\forall j)[\mathscr{C}^1(W_j) \text{ is beefy} \Rightarrow (\exists f \in \mathscr{C}^1(W_j))[MinProg(f) \leqslant h(MinProg_n(f))]]$.

Proof. Fix n. To facilitate the definition of h, we first introduce, for each $i \in N$, m_i. If there exists a recursive function $g \in \mathscr{C}^1(W_i)$ such that $MinProg_n(g) > i$, then $m_i = MinProg(g)$; $m_i = 0$ otherwise. We now define h:

$$h(i) = \begin{cases} m_0 & \text{if } i = 0; \\ 1 + max(\{m_i, h(i-1)\}) & \text{otherwise.} \end{cases}$$

It is easy to see that h is increasing and, for some j, if $\mathscr{C}^1(W_j)$ is beefy, then there exists a $g \in \mathscr{C}^1(W_j)$ such that $MinProg_n(g) > j$.

We now show that, for each j such that $\mathscr{C}^1(W_j)$ is beefy, there exists an $f \in \mathscr{C}^1(W_j)$ such that $MinProg(f) \leqslant h(MinProg_n(f))$. We take $f = \varphi_{m_j}$. Clearly, $m_j = MinProg(f) \leqslant h(j)$. But, since h is an increasing function and $j < MinProg_n(f)$, we have that $MinProg(f) \leqslant h(MinProg_n(f))$.

A straightforward Tarski–Kuratowski [21] quantifier analysis of our construction in the proof of Theorem 4 just above yields that the h presented there can be represented as *three* iterated limits of a computable function. We suspect, but did not check that a modified construction (if not the one given) would yield an h representable as *two* iterated limits of a computable function.

We now present the general result from which Corollary 2 follows. This theorem presented next shows that given a *limiting recursive* h, for any $n \in N$ and an index, i, of a program which computes h in the limit, we can algorithmically obtain, via a recursive function g, an r.e. index $g(i,n)$ for a class of partial recursive functions whose 1-extension is beefy and there exist n machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n$ such that for $1 \leqslant k \leqslant n$, machine $\mathbf{M}_k$ $\mathbf{Ex}^k$-identifies **Fin**, and for any $k < n$, machine $\mathbf{M}_{k+1}$ $\mathbf{Ex}^{k+1}$-identifies each $f \in \mathscr{C}^1(W_{g(i,n)})$ h-more succinctly than any machine that $\mathbf{Ex}^k$-identifies each $f \in \mathscr{C}^1(W_{g(i,n)})$.

Theorem 5. $(\exists g \in \mathscr{R}^2)(\forall n > 0)(\forall i, h \mid h = \lambda x.\lim_{s \to \infty} \varphi_i(x,s) \text{ is total})[[\mathscr{C}^1(W_{g(i,n)}) \text{ is beefy}] \wedge [(\exists \mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n)[(\forall k \mid 1 \leqslant k \leqslant n)[\mathbf{Fin} \subseteq \mathbf{Ex}^k(\mathbf{M}_k)] \wedge (\forall f \in \mathscr{C}^1(W_{g(i,n)}))(\forall k < n)[MinProg_k(f) > h(\mathbf{M}_{k+1}(f))]]]]$.

Proof. Suppose i and n are given. By the s-m-n theorem [21], we will construct a recursive function p. It will be easy to see that, a program for p can be found algorithmically in i,n. We define $W_{g(i,n)} = \{p(j) \mid j \in N\}$.

For $1 \leqslant j \leqslant n$, let $\mathbf{M}_j$ be the machine described in the proof of Theorem 2. These same machines serve our purpose in the present proof. Note that machine $\mathbf{M}_j$ $\mathbf{Ex}^j$-identifies **Fin**.

For $k \in N$, we give, below, an informal description of program $p(k)$. According to our construction, $\varphi_{p(k)}$ will converge on all except n arguments. We will define $\varphi_{p(k)}$ in stages.

begin {Definition of $\varphi_{p(k)}$}
  begin {Initialization}
    1: for $x \leqslant k$, let $\varphi_{p(k)}(x) = 1$;
      {Note that this ensures that the class of functions $\mathscr{C}^1(W_{g(i,n)})$ is beefy.}
    2: for $1 \leqslant j \leqslant n$, let $a_j = k + j$;
      {We will use a_j as moving markers. We will argue in Claim 1 that the a_j's eventually reach final values. These final a_j's will be the only points at which $\varphi_{p(k)}$ is undefined.}
    3: let $y = k + n + 1$;
    4: for all $r \in N$, let $Injure(r) = 0$;
      {To facilitate seeing that this initialization of each $Injure(r)$ preserves algorithmicity, we note that, clearly, each individual initialization could be carried out instead just before its first use.}
    5: go to Stage 0;
  end {Initialization}

  begin {*Stage s*}
    6: **if** there exist j and r such that the following hold:
      6.1: $j < n$;
      6.2: $r \leqslant \varphi_i(Prog([0..k] \cup \{a_l \mid 1 \leqslant l \leqslant j\}), s)$;
      6.3: $Injure(r) < n$;
      6.4: $\varphi_r(a_m)\downarrow = 1$, for some $m > j$, *and* in at most s steps
    then
      6.5: let j and r be one such pair;
      6.6: let $Injure(r) = Injure(r) + 1$;
      6.7: for $a_{j+1} \leqslant x \leqslant y$, let $\varphi_{p(k)}(x) = 0$;
      6.8: for $j < m \leqslant n$, let $a_m = y + m - j$;
      6.9: let $y = y + n - j + 1$
    else
      6.10: let $\varphi_{p(k)}(y) = 0$;
      6.11: let $y = y + 1$
    endif;
    7: go to Stage $s + 1$
  end {*Stage s*}
end {*Definition of* $\varphi_{p(k)}$}

As mentioned before, we let $W_{g(i,n)} = \{p(j) \mid j \in N\}$. Note that an index for p can be algorithmically obtained from i and n.

For the rest of the proof we suppose i and n satisfy the hypotheses of Theorem 5. The theorem follows from Claims 1–3.

Claim 1. $(\forall k)(\forall l \mid 1 \leqslant l \leqslant n)[a_l$ *reaches a final value for program* $p(k)]$.

Proof. We give a proof by induction. We begin by showing that a_1 reaches a final value. Let s_1 be the least Stage in program $p(k)$ such that the following two conditions hold:

(1) $(\forall s \geqslant s_1)[\varphi_i(Prog([0..k]), s) = \varphi_i(Prog([0..k]), s_1)]$;

(2) $(\forall r \leqslant \varphi_i(Prog([0..k]), s_1))[Injure(r)$ has reached its maximum value].

Clearly, such an s_1 exists because φ_i is limiting recursive and condition 6.3 in the description of program $p(k)$ guarantees that, for *no* $r \in N$, does $Injure(r)$ become greater than n. Now, the only way in which a_1 can move after Stage s_1 is by selection of $j = 0$ and some $r \leqslant \varphi_i(Prog([0..k]), s)$ in step 6 of program $p(k)$. But, such a situation would cause the value of $Injure(r)$ to increase contradicting the fact that $Injure(r)$ has already reached its maximum value.

Now, suppose, by induction, that for some $m < n$, $a_1, a_2, \ldots, a_m$ reach a final value. Using an argument similar to the a_1 case, we can show, from the induction hypothesis, that a_{m+1} reaches a final value. □

For the rest of the proof we take $a_1, a_2, \ldots, a_n$ to be their *final* values.

Claim 2. *For each* $k \in N$,

$$\varphi_{p(k)}(x) = \begin{cases} 1 & if x \leqslant k; \\ \uparrow & if x \in \{a_1, a_2, \cdots, a_n\}; \\ 0 & otherwise. \end{cases}$$

Proof. Clear from the construction of program $p(k)$. □

Claim 3. $(\forall k)(\forall j < n)(\forall r \leqslant \lim_{s \to \infty} \varphi_i(Prog[0..k] \cup \{a_m \mid m \leqslant j\}), s))[\varphi$*-program* r *makes at least* $n - j$ *errors in computing* $\chi_{([0..k] \cup \{a_m \mid m \leqslant n\})}]$.

Proof. If the final value of $Injure(r)$ is $\geqslant n - j$, we are done because then φ-program r explicitly makes $n - j$ errors computing $\chi_{([0..k] \cup \{a_m \mid m \leqslant n\})}$. Therefore, we suppose the final value of $Injure(r)$ is $< n - j$. Now, for each $n - j$ markers $a_{j+1}, a_{j+2}, \ldots, a_n$, φ-program r does not converge to 1. But $\chi_{([0..k] \cup \{a_m \mid m \leqslant n\})}(x) = 1$ for $x \in \{a_{j+1}, a_{j+2}, \ldots, a_n\}$. The claim follows.

This completes the proof of Theorem 5. Of course, because of Theorem 4, we cannot obtain such results for completely arbitrary h. □

We considered beefy collections to ensure that our results were true for nontrivial classes. Finally, we would like to note that our results can also be shown to hold for a slightly modified notion of beefiness in which we require that for each pair of disjoint finite sets D' and D'', the class $\mathscr{C}$ contain the characteristic function of a finite set D such that $D' \subseteq D$ and $D'' \subseteq \overline{D}$. This notion is introduced below.

Definition 9. $\mathscr{C}$ is *strongly beefy* $\iff$ $[[\mathscr{C} \subseteq \mathbf{Fin}] \wedge [(\forall D', D'' \text{ finite} \mid D' \cap D'' = \emptyset)(\exists D \text{ finite})[D' \subseteq D \wedge D'' \subseteq \overline{D} \wedge \chi_D \in \mathscr{C}]]]$.

It is straightforward to show that analogs of Theorems 2 and 5 hold for strongly beefy classes.

Acknowledgements

The research was supported in part by NSF grant CCR 832-0136 at the University of Rochester, NSF grant CCR 871-3846 at the University at Delaware, a grant from the Siemen's Corporation at MIT, and an Australian Research Council grant at UNSW. We would also like to express our gratitude to Professor S.N. Maheshwari of IIT, Delhi for making the facilities of his department available to us. Finally, we would like to thank the referees for many helpful suggestions.

References

[1] M. Blum, A machine independent theory of the complexity of recursive functions, *J. ACM* **14** (1967) 322–336.
[2] M. Blum, On the size of machines, *Inform. Control* **11** (1967) 257–265.
[3] L. Blum and M. Blum, Toward a mathematical theory of inductive inference, *Inform. Control* **28** (1975) 125–155.
[4] J. Case, S. Jain and A. Sharma, Complexity issues for vacillatory function identification, *Inform. Comput.* **116** (1995) 174–192.
[5] J. Case, S. Jain and M. Suraj, Not-so-nearly-minimal-size program inference, Tech. Report 94-25, Univ. of Delaware, Newark, Delaware, 1994.
[6] J. Case and C. Smith, Comparison of identification criteria for machine inductive inference, *Theoret. Comput. Sci.* **25** (1983) 193–220.
[7] K. Chen, Tradeoffs in machine inductive inference, Ph.D. Thesis, SUNY at Buffalo, 1981.
[8] K. Chen, Tradeoffs in inductive inference of nearly minimal sized programs, *Inform. Control* **52** (1982) 68–86.
[9] A. Church, *The Calculi of Lambda Conversion* (Princeton Univ. Press, Princeton, 1941).
[10] R. Freivalds, Minimal Gödel numbers and their identification in the limit, Lecture Notes in Computer Science, Vol. 32 (Springer, Berlin, 1975) 219–225.
[11] R. Freivalds and E. B. Kinber, Limit identification of minimal Gödel numbers, *Theory of Algorithms and Programs 3;Riga 1977*, (1977) 3–34.
[12] M.A. Fulk and S. Jain, Approximate inference and scientific method, *Inform. Comput.* **114** (1994) 179–191.
[13] E.M. Gold, Language identification in the limit, *Inform. Control* **10** (1967) 447–474.
[14] S. Jain and A. Sharma, Program size restrictions in computational learning, *Theoret. Comput. Sci. A* **127** (1994) 351–386.
[15] E.B. Kinber, A note on limit identification of c-minimal indices, *Electronische Inform. Kybernetik* **19** (1983) 459–463.
[16] M. Li and P. Vitányi, An introduction to Kolmogorov complexity and its applications, Texts and Monographs in Computer Science (Springer, Berlin, 1993).
[17] M. Machtey and P. Young, *An Introduction to the General Theory of Algorithms* (North-Holland, New York, 1978).
[18] A. Meyer, Program size in restricted programming languages, *Inform. Control* **21** (1972) 382–394.
[19] A. Meyer and A. Bagchi, Program size and economy of description, in: *Symp. on the Theory of Computation* (1972).

[20] H. Rogers, Gödel numberings of partial recursive functions, *J. Symbolic Logic* **23** (1958) 331–341.
[21] H. Rogers, *Theory of Recursive Functions and Effective Computability* (McGraw Hill, New York, 1967, Reprinted, MIT Press, Cambridge, 1987).
[22] J. Royer, Inductive inference of approximations, *Inform. Control* **70** (1986) 156–178.
[23] J. Royer and J. Case, Progressions of relatively succinct programs in subrecursive hierarchies, Tech. Report 86-007, Computer Science Department, University of Chicago, 1986.
[24] J. Royer and J. Case, *Subrecursive Programming Systems: Complexity and Succinctness*, Progress in Theoretical Computer Science (Birkhäuser Boston, 1994).
[25] N. Shapiro, Review of "Limiting recursion" by E.M. Gold and "Trial and error predicates and the solution to a problem of Mostowski" by H. Putnam, *J. Symbolic Logic* **36** (1971) 342.
[26] J. Shoenfield, On degrees of unsolvability, *Ann. Math.* **69** (1959) 644–653.
[27] J. Shoenfield, *Degrees of Unsolvability* (North-Holland, Amsterdam, 1971).
[28] R. Soare, *Recursively Enumerable Sets and Degrees* (Springer ,Berlin, 1987).